6

D4

G06F 15/16B4N
H04L 12/24

XP 000207780

8302 IEEE Network
5(1991)March, No.2, New York, US

NETMATE: A Network Management Environment

E

p. 35-40,43

Alexander Dupuy
Soumitra Sengupta
Ouri Wolfson
Yechiam Yemini

As computer networks become larger (often involving thousands of elements), more heterogeneous (supporting a variety of devices and protocols), and more complex (involving subtle interactions and relationships among components), the importance of network management for large-scale systems increases. For the network to be the information highway for an enterprise, there must exist effective management tools to ensure smooth and efficient operations.

A typical enterprise-wide network is managed today by a set of nonintegrated tools, each of which has a partial and incomplete view of the network and its management needs. This leads to significant difficulties in accomplishing effective management (see [1] for sample problems). The prime goal of the Network Management, Analysis, and Testing Environment (NETMATE) project is to develop a unified and comprehensive software environment for network management to oversee and orchestrate the operations of diverse devices and protocols.

A key component of a network management environment is its model of the network. This model presents to management tools the knowledge available to the management system. The organization of such a model and its architectural relationship with the management tools are the central theme of this article. The problem of effective network modeling gives rise to difficult technical challenges. The network objects whose operational behavior must be captured by the model are typically distributed. For example, a virtual circuit in an X.25 network may involve a set of distributed processes and data structures maintained by multiple switches and network controllers. The state of such a distributed object may be only partially observed by each individual device that supports it. Reports of such observations to a central modeling facility may be inconsistent, e.g., one switch believes the circuit is lost while another believes it is operating. A modeling facility may thus have to handle intrinsically inconsistent data. How can concerted management action be achieved in the presence of such inconsistencies?

Similarly, the behaviors of objects in the network may be highly correlated. In a multilayered protocol, the behavior of an entity in one layer may be implemented in terms of a set of entities in another layer. For example, in the Transmission Control Protocol/Internet Protocol (TCP/IP) suite, a Telnet virtual terminal session depends on the well-being of transport-layer (TCP) and network-layer (IP) functions. A failure of the network-level connections may lead to correlated failures in the higher layers. This mapping of layered objects must be represented by the network model to allow interpretation of correlated behaviors. However, not all behavior correlations can be captured by static configurational relations. Some correlations may result from dynamic interactions among objects. For example, the noise levels associated with a physical link may be correlated with the rate of control packets sent by higher layers. It is necessary for the model to represent such dynamic relations as well as more static relations. How can a model represent a spectrum of correlations among object behaviors?

Another problem is that of many different protocols running on a single network, with intricate dependencies between each other. For example, it is common for an enterprise to have both Ethernet and token ring with protocols such as TCP/IP and DECNet over Ethernet and TCP/IP and Systems Network Architecture (SNA) over token ring operating concurrently. Existing protocol-specific network management systems typically have models that reflect only the properties and relationships supported within their protocols, and cannot support the analysis of interdependency information.

The NETMATE project is building a collection of tools for comprehensive network management in a distributed environment, addressing management problems such as those described above. The overall design emphasizes a distributed approach: the definition and collection of network information, and the storage, analysis, and review of this information, may

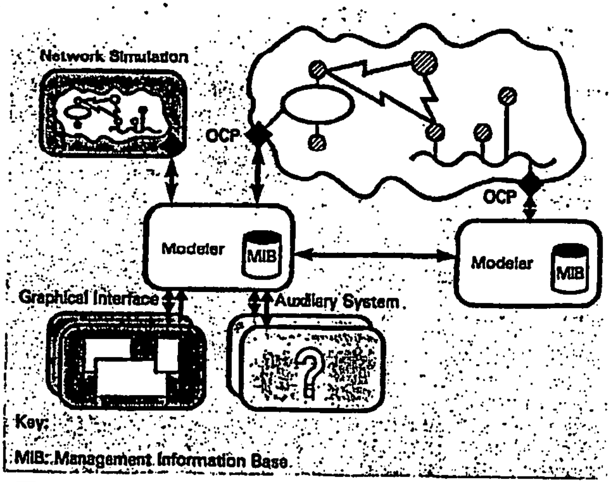

Fig. 1. NETMATE architecture.

Research supported by Defense Advanced Research Projects Agency (DARPA) contract #F-29601-87-C-0074 and NY State CAT contract #NYSSTF CAT(89)-5. Opinions expressed herein are strictly those of the authors.

0890-8044/91/0003-0035 $01.00 © 1991 IEEE

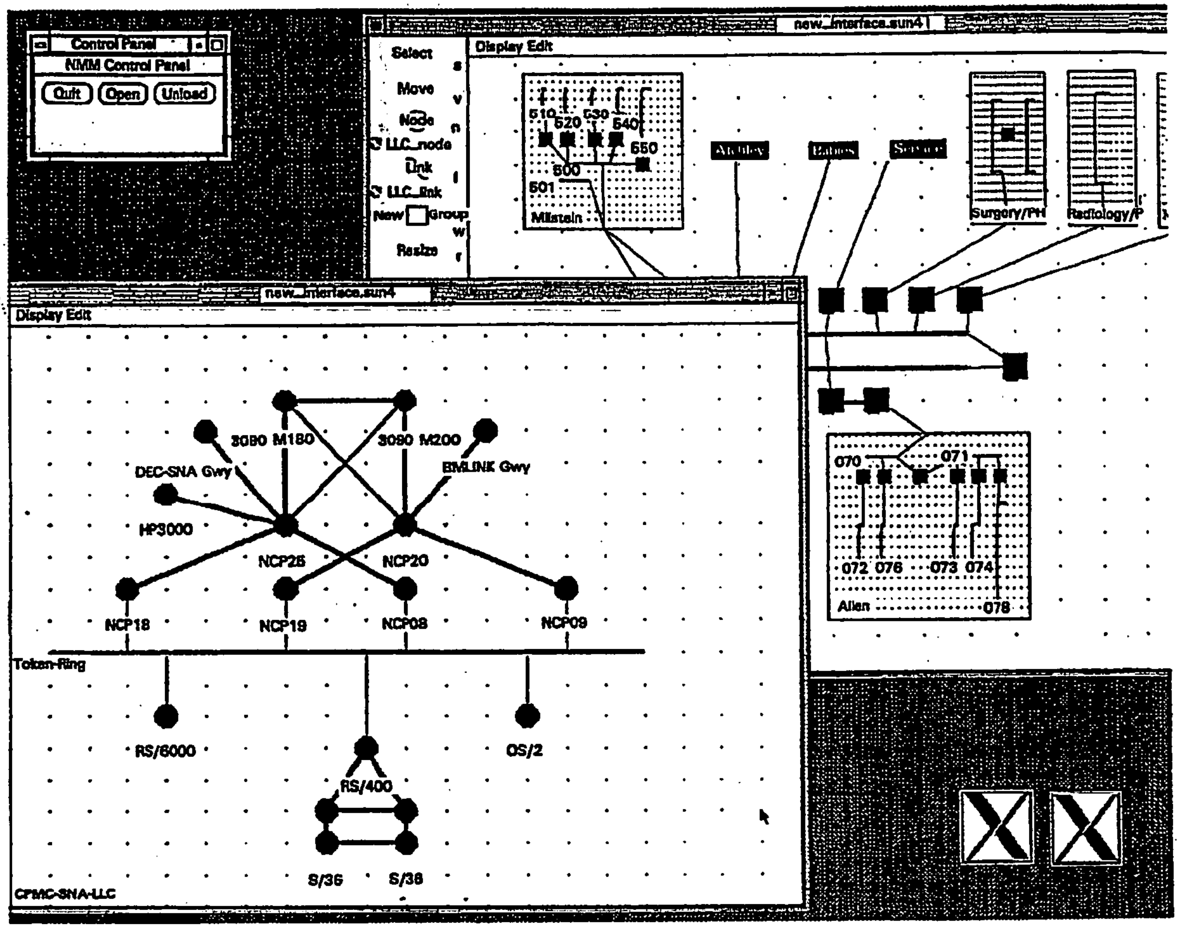

Fig. 2. UI component.

occur at several locations. The central theme of NETMATE is the design and implementation of a generic data model to represent network entities and their interrelationships. The model is extensible as it permits the addition of new protocols and entities with different sets of properties and analysis procedures. These tools, developed around this generic network model concept, address a varied set of problems in day-to-day network management.

The rest of the article is organized as follows. The overall NETMATE architecture is discussed next. The problem of network modeling and the NETMATE approach to it are then presented. The current implementation status of NETMATE is given, and conclusions are in the final section.

Overall NETMATE Architecture

The NETMATE architecture is a collection of network management tools. The major components of a complete NETMATE environment are the Modeler, User Interface (UI), Observation/Control Point (OCP), Simulator, and auxiliary systems (see Figure 1). In the following, we describe the network management functions performed by each component.

Modeler

The NETMATE Modeler component supports the representation of complex network models in order to permit efficient interpretation and response to problems. Additionally, it allows other NETMATE components to communicate with each other by serving as the hub of the system. As the primary repository of information, it connects the other components together. All other components function as clients of the Modeler. In a distributed implementation, the Modeler also provides services to other NETMATE environments in the system and takes part in the coordination of information flow with other Modelers. The Modeler is implemented using a database, so that it is easy to provide database facilities like concurrency control and recovery to the clients. The "Network Model" section will describe, in detail, the NETMATE approach to network information modeling as implemented by the Modeler.

User Interface

The UI component provides the ability to visually navigate and control complex network scenarios. The UI maps numerous objects and relationships into a visual representation for a human network manager. Furthermore, due to limitations of visual displays, the UI supports multiple visual abstractions to reduce clutter and focus on the entities of interest.

A simple but important feature is that a user may see multiple views of the network simultaneously. The UI allows the network to be viewed at multiple layers, e.g., the IP network layer or SNA link layer. Consider a network consisting of Local Area Networks (LANs) (Ethernet and token ring) with multiple higher-layer protocols (TCP/IP and SNA) (see Figure 2). During a problem-determination scenario in the SNA network, the user may begin with the SNA view,[1] which is depicted in the bottom-left window in the figure. Assume that a problem is determined in an SNA link between the nodes NCP18 and AS/400, which maps to a token ring. The token ring network view may then be made visible, as is depicted in the top-right window. The views not of interest may be iconized (bottom right).

An additional technique used to reduce clutter is the hierarchical display concept in the UI. As the token ring network view is shown to the user, all visual groups within the view that are not of interest (i.e., the entities that are not part of the SNA link having the problem) may be displayed in an iconized form (e.g., the three smaller squares at the top of the token ring view). The decision of which groups to make visible depends on the policy employed in the UI to map network model information to the visual model information. By using the facilities of the control panel and the menu choices, the user may decide to take further actions to resolve the problem.

Observation and Control Point

In NETMATE, access to the objects in the network is provided by OCPs. An OCP is a management agent, and it uses manager-agent protocols to support communication. An OCP functions as a filter to reduce the amount of information and updates being sent to the Modeler, as well as perform translation functions between the native management protocols used by devices and the Modeler interface. A generic OCP of the type described in [2] can handle dynamic filtering and translation criteria as required by managers, using software and protocol structures that permit managers to delegate management programs to agents.

The separation of OCP modules and functions from the Modeler is significant in two ways. First, by keeping the interface between the NETMATE system and other network management systems in a separate component, it is possible to avoid having many different device-dependent and protocol-specific routines in the Modeler. Second, it allows a convenient addition of interfaces from the Modeler to new networks. The only protocol the Modeler has to follow is the manager-agent protocol.

An important feature of the manager-agent protocol allows the Modeler to instruct an OCP to send information to the Modeler when a specific event occurs. This instruction is conveyed by delegating a management program to the OCP. For example, the program may return status information of a basically static network entity only when its value changes; or when it changes beyond some threshold. Another example of a management program is one that converts polling of devices into alerts and vice versa. If a network device is not intelligent enough to generate alerts when problems exist, a management program can poll that device and send an alert to the Modeler when it detects a problem. On the other hand, a device may generate many alerts that are of no interest unless there is some other problem indication. In this case, a management program could just store the alerts, and would be polled by the Modeler whenever other problems were detected.

Simulator

NETMATE architecture includes a simulation component for building "what-if" analysis capabilities into a network management system. It addresses the problem of incorporating simulated scenarios with management capabilities as a means to examine and debug planned network enhancements, and also for operator training. This feature is also useful in network design and research in network protocols and distributed systems [3].

A typical use of the simulation capability under a "what-if" scenario is illustrated by this example: Take a snapshot of the current network state from the Modeler, load it into the Simulator, alter the network configuration, and observe the effects in the simulated network. Temporal information stored in the Modeler could be particularly useful for developing models of the various network components for the simulation. This scenario is just one case of a more general use for simulation as a bridge between network management on the one hand, and network design and capacity planning on the other.

From a network simulation perspective, integration with a network management system is also useful. In a typical scenario for network protocol research, the features of UIs and analysis tools developed for network management are exactly what is needed for getting information such as packet counts and rates, traffic levels, and loads out of a network simulation and presenting it to a researcher in an easily understood manner.

Auxiliary Systems

Auxiliary systems can use the network database services of the Modeler to support other related activities, such as inventory and trouble ticketing. As other system management capabilities become available, such as managing user accounts, storage devices, and file systems, we will use the NETMATE model to manage these as well.

NETMATE maintains information useful to other systems. For example, the configuration database can be used for inventory control, tracking specific hardware and software systems in the network. If vendor and price information is added, such a comprehensive system may be used by the purchasing department. With such information, when a problem is recognized with some equipment, quick servicing or reordering actions can be carried out.

NETMATE includes a Trouble Ticketing system. This system is used for recording network problems and their handling. Typically, a ticket is created for every reported trouble, and its status changes as it undergoes investigation and is eventually resolved. The Modeler maintains information about long-lasting network problems. The Trouble Ticketing system is also used as a reference facility. When a problem is reported, old trouble tickets are queried to suggest a course of action. It is designed to identify problems that are more frequent and hence need special attention.

Network Model

In discussing the network model of NETMATE, we will use a network example to examine a number of network management scenarios and how the model would support management operations. Consider a link-layer network of an Ethernet (*E*) with two nodes (A and B) and a token ring (*T*) with one node (D) (see Figure 3). *E* and *T* are connected by a bridge (C). The Ethernet *E* is divided into two segments joined by a repeater (R). Additionally, C is also connected to a serial line (*S*). Using this simple network as an example, we will elaborate a few nontrivial problems associated with network modeling, and present NETMATE solutions with qualitative comparisons of the NETMATE model with other standard models.

Consider a network problem where B is unable to communicate with D. An analysis process may attempt to construct all possible connection paths from B to D, and test components in each path to determine the problem. The program may discov-

---

[1] Since NETMATE does not dictate a layer to be defined based on any existing network protocol layer, it is possible, for example, to model all SNA entities to be in a single layer, and hence a single view.

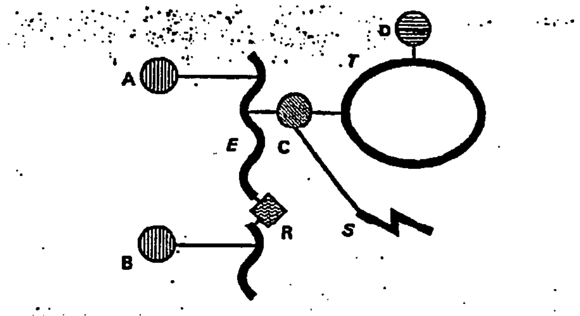

Fig. 3. Network example.

er that these nodes are on different types of links, and then attempt to find a bridge (among all bridges in the network) that may have both nodes in its forwarding table property.[2] Using only the information from the nodes (assuming they are reachable from the management system), it will have to algorithmically deduce which Ethernet and token ring segments are part of the connection between B and D to construct the connection paths. This process becomes more complex if the nodes are separated by more than one bridge. Also note that the analysis program has to understand individual nodes' vendor- and protocol-specific properties to construct the paths.

The process is simplified if the model allows representation of generic object classes, node and link, and a generic connection relationship between instances of these classes. The relation in the model is maintained as nodes get connected to the links in the network. Thus, the connection relation at the beginning of the analysis has the information that *E* is connected to A, B, and C, and *T* to C and D. Given such a model, it is fairly easy to use a graph-topological algorithm to construct the connection path B-*E*-C-*T*-D before testing any individual component in the network. Furthermore, since it is also known that A is connected to *E*, A may be directly queried to check the status of *E*, which is not possible in the International Organization for Standardization (ISO) model unless it has been previously deduced that A and B are on the same link *E*. Also note that with this information, the path-searching algorithm need only look at the connection relation and not at any protocol-specific properties of the nodes. The same information may be used to easily create a display of the network, as shown in Figure 3.

The NETMATE model, based on object-oriented database technology, includes the generic node and link object classes and the connection relation (see Figure 4) in its class hierarchy. Note that links in the network are not active components; the model, nevertheless, distinguishes different properties between nodes and links, and makes it explicit through class definitions. The NETMATE UI module makes use of the connection relation in displaying network objects.

Assume that in the network example, the Ethernet segment that connects to B, physically has a repeater R due to distance limitations. The analysis process, upon querying A, determines that there exists a path from A to D. The problem, thus, is in the Ethernet segment that connects B, or that segment's physical cable connection to B, or at B itself. To investigate further, the process needs to know the corresponding physical-layer information, i.e., that the Ethernet in the link layer is implemented in terms of physical-layer network objects such as repeaters, segments, and hubs. Note that this investigation of the physical layer is unnecessary until the time the fault is determined to be in a specific link layer object.

In general, in almost all protocols there is a layering concept in which a network component at one layer is implemented in terms of a set of network components at another layer. This basic concept is represented in NETMATE using the layer object class and the mapping relation. Given this generic information, the analysis process may continue investigation of objects that a problem object maps to at another layer. Once again, although the mapping between objects is determined by the specific nature of individual protocols, the analysis process is generic in that it needs to know only the mapping relation. Note that in NETMATE, a layer is independent of a specific protocol suite, and there is no preconceived notion of a "higher" or "lower" layer. This is useful in scenarios where in some part of the network DECNet runs over IP and, in another part, IP over DECNet.

Consider an example where the communication problem occurs at C due to unavailability of memory buffers for the bridging of *E* and *T*. Having detected the problem at C, the analysis process needs to examine the traffic behavior on all individual interfaces in C, including the one to *S*, since sum-total use of buffers at C depends on all three interfaces. The generic modeling need, therefore, is to be able to represent subobjects of an object by the relation components. This allows C to have three subnode components, each with an independent set of properties contributing to the sum-total properties of C. Additionally, the NETMATE semantics to this relation are that while a subnode has all the same properties as a generic node, it cannot exist independent of its supernode. The component concept is also useful for link objects, such as for modeling of SNA Multidrop lines, and SNA Twinax connections.

Consider an example where the token ring has many nodes that communicate with B (perhaps an application server), and C is inoperative. When invoked as a result of D-to-B communication failure, if the analysis process determines the cause, it need not perform the analysis again for any other similar token ring node. This correlation may be achieved by grouping such a set of network objects under a common criterion. In general, such grouping needs arise due to many reasons, such as organizational hierarchy, topological or geographical proximity, and reporting and analysis processes. This is represented in NETMATE by the group object class and the grouping relationship. Groups are also used by the UI module to create visual groups on the display.

Consider an example of a problem that is specific to a vendor implementation of Ethernet connections for A and B. In a heterogeneous environment, it is not possible for an analysis process to always come up with a solution and corrective actions based only upon generic information. The model must therefore accommodate specific information and testing and corrective processes, which may be invoked by analysis procedures once the management context is determined. This is achieved in NETMATE by user-defined types, methods, and instances that inherit the generic properties and relationships of NETMATE object classes. These definitions are supported in NETMATE as additional network-specific-type hierarchies under the basic object classes (e.g., the TCP virtual circuit class in Figure 4).

The existing two standard data models, known as Structure of Management Information (SMI), are the Internet SMI [4] and the ISO SMI [5]. These standards provide a common structure for management data for heterogeneous networks, and allow a mechanism for the definition and naming of variables containing management information (essentially, name-value pairs). Some additional structuring is provided; tables of variables can be defined (although the Internet SMI does not support nested tables). The ISO SMI has an object-oriented model,

---

[2] This information may not be found if the bridge uses an adaptive algorithm, and the entries are deleted due to timeout.

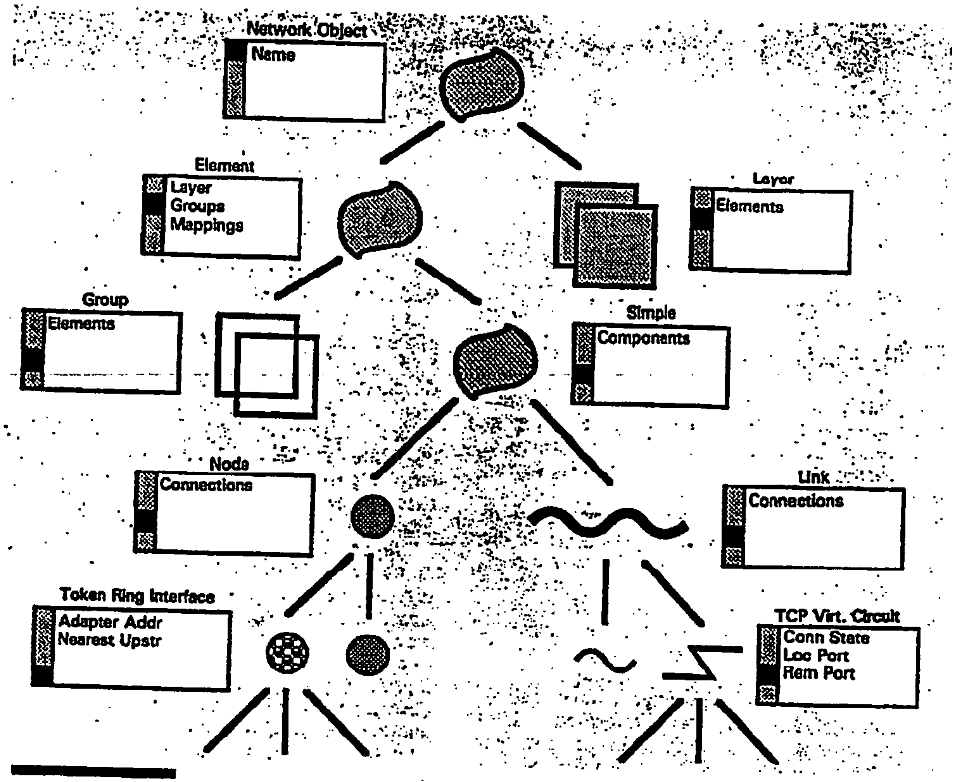

Fig. 4. Network model class hierarchy.

with variables specified as attributes of objects, an inheritance mechanism for defining object classes, and a single containment relationship between objects.

In the ISO model, which has a single top-level generic object class, each node is represented as an instance of a protocol-specific subclass of the top class. Therefore, A will be an instance that has properties related to its Ethernet interface, whereas an instance for D will have properties related to its token ring interface. Since the links are not directly represented in the ISO model, it does not directly keep any information about them. The fact that there exists an Ethernet, and that two nodes are connected to the same Ethernet, will have to be algorithmically deduced, at a great cost, from the properties of individual nodes every time this network needs to be displayed or analyzed.

A significant number of network queries (model accesses) are likely to access the fundamental relations mentioned above. Thus, if the model fails to provide a sufficiently rich set of these fundamental relations, applications will incur the significant unnecessary cost of constructing these relations. The Internet and ISO SMI models lack such relationships (specifically mapping), which are essential for efficient automated network problem analysis.

A network management system must support analysis of collected information to recommend specific actions; in other words, it must not only define and collect, but also manage the information. The model, therefore, must allow for specification of network management rules that are applicable on specific network objects. Based on information in the network model, these rules are triggered for further analysis. For example, `modem_repair(M) :- modem(M), (M.status != "OK"), reset(M).` is a simple rule for the Modem class (where *M* is an object instance for, say, a modem on *S*). Complex analysis of network objects may be performed as procedures with decision systems such as neural network and queuing analysis. These are necessary when simple deterministic steps are not sufficient. For example, an extensive queuing and pattern analysis correlation may show that certain parameters such as `Modem.Speed` and `Modem.PacketSize` need to be dynamically reassigned over time to improve throughput.

Information about previous values of management data, and explicitly time-related information such as traces and data series, need to be provided for. A trend analysis for modem throughput, for example, requires availability of data such as `Modem.Speed`, `Modem.PacketSize`, and `Modem.NumPackets` at frequent intervals of time. Furthermore, the network being a real-time system, careful distinctions need to be made between the real time when such values are sampled and the time when such values are stored in the model.

Further research issues and detailed descriptions for both rules and procedures and temporal specification of values and events in NETMATE can be found in [6].

Distributed Management Issues

This section examines the research problems in distributed network management. First, since network information is distributed over network components, the management function is inherently distributed. Second, this information is collected in the model for management purposes, but the management architecture consists of distributed components, and thus the management protocol adds another layer, which may be called distributed network management.

The primary problem with distributed network information is that of the reliability or correctness of that information. For example, for passive network components such as links, it

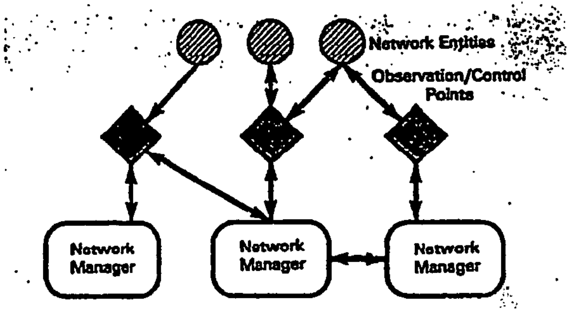

Fig. 5. Distributed management model.

is quite possible that a node connected to the link determines the link to be operating, and another node, unable to connect to the link, determines it to be inoperative. While the information seems to be inconsistent, both reports are true from the partial viewpoint of the observing nodes.

If the network model is considered simply a database with serializability as the notion of correctness, the data in the model may not accurately reflect the network being modeled. For example, if the status of the link is updated simply by the last report from any node on the link, it is quite possible to have information in the model contrary to the real network, serializability notwithstanding. The reason is that while traditional database concepts impose the tradition that a transaction (typically written by humans) is assumed to be consistent, such an assumption is invalid in a network information database, where a transaction is an update of information based on values retrieved from the network. Short of freezing all network activities, the information in the model will always be out of date with the real network information. Under such conditions, the consistency and timeliness of the network model may perhaps be attributed with only a "degree of confidence." Policies—such as "a link is inoperative only if majority of nodes report it to be inoperative for a duration of 10 s"—are examples of heuristic reasoning that may be applicable.

Another issue in distributed management relates to possible network degradation caused by the management system itself. Consider a very large network with a single manager; excessive network management traffic would adversely affect both the network's and manager's performance. In NETMATE, this problem is addressed by defining a management architecture consisting of elements in three levels: first, the network devices themselves; second, simple managing agents (OCPs), which support the native device management protocol and the protocol used by the distributed management system; and third, the managers themselves (see Figure 5). As mentioned previously, the ideal management protocol supported by these management entities is an active research topic [2]. The architecture, however, raises three other issues—access, concurrency, and replication control—each with a special problem in the network management context.

Access control problems relate to deciding which managing entities have rights to control which OCPs and devices. Ultimately, if access control is not supported by the network devices, providing it at the other levels is pointless. However, the type of access control supported by network devices can be very primitive ("all-or-nothing"), and therefore, access control has to be built within the management structure of managers and OCPs.

Concurrency control, on the other hand, may not be desirable to implement in network devices, since doing so might prevent the ability to bypass the distributed management system when it is necessary to correct a problem quickly. Additionally, the complexity of supporting transactions in network devices is impractical. However, at the manager level, where it will be quite common to have full database management systems, concurrency control and transaction management support will often be available at little incremental cost. Nevertheless, in some cases, allowing concurrent management access to the network may not be desirable, as there are usually no ways to back out from some actions such as "reboot."

When determining whether management information should be replicated in more than one layer (or across Modelers), a tradeoff is made between providing the most current and accurate information and reducing the traffic and load on network elements from frequent requests for the information. Additionally, replicated information may act as a cache to speed management applications. Other issues include how redundancy should be provided and how to configure the distributed system (e.g., network devices may not have sufficient intelligence to choose primary OCPs/managers, let alone backup OCPs/managers).

While the OSI and Internet architectures address these issues in part (mostly regarding access control mechanisms), they do not address concurrency control, and other issues are dealt with incompletely.

Since networks are thoroughly interconnected, a high-level application may communicate with another application at a distant node and cut across many management domain boundaries. In such cases, spill-over problems can occur in many ways: a problem in one domain may prohibit regular functions in other domains (for example, file transfer to a remote node fails at a regional network because the long distance network is down), or a problem in one domain causes active disorder in another domain (for example, TCP/IP networks may be flooded with unnecessary address resolution protocol traffic due to improper configurations). A cooperative management strategy needs to be developed for distributed problems that span over management domains on interconnected networks.

The approach taken in the NETMATE system is to provide policies for structuring distributed management systems. For example, by limiting OCPs to communicate only with a single manager, access and concurrency control problems at that layer are substantially simplified. Also, having the OCPs initiate the connection to their manager, rather than vice versa, nearly eliminates the need for access control entirely. On the other hand, replicating data in the OCP can improve performance significantly without sacrificing much accuracy in the information. These issues form future research topics in NETMATE.

NETMATE Status

Currently, a prototype NETMATE system has been developed at Columbia with a functional Modeler, UI, and OCP. The Modeler is implemented using the Vbase object-oriented database running on Sun-3 machines; the UI is written in C++ using the Interviews X window system toolkit from Stanford. One OCP has been developed, with a generic interface using the Simple Network Management Protocol (SNMP) to communicate with managed network objects. Both the UI and OCP are currently running on Sun-4 machines.

The development of the UI is ongoing, and enhancements and additional features are planned. A new OCP, written in C++, which will provide more of the data transformation and reduction capabilities mentioned above, is under development. A Modeler with object-oriented structures based on a relational database is planned, and implementation should begin soon. A trouble-ticketing application will be written as this new Modeler becomes usable. The Nest network simulation tool developed at Columbia [3] will be used as the basis of a Simulator component.

The existing and planned components of the NETMATE system are being used as a platform for research by project members in a number of areas: concepts for distributed management with access control, automated analysis and network management functions, filtering and storage of variable data, and a generic network management information exchange language.

Conclusion

NETMATE addresses the management needs of network administrators by supporting a generic data model and a comprehensive set of distributed tools. The model exploits generic properties and relationships exhibited by network components in most networks in order to facilitate efficient analysis methods. The model is both comprehensive and extensible. Many complex issues related to distributed management are integrated into the design and research of the model architecture. The self-contained tools (e.g., UI, OCP, analysis modules, and trouble ticketing) encapsulate specific functions and lend themselves to a modular and efficient network management solution.

References

[1] G. Barzilai *et al.*, "Network Fault Management: A User's View," *Proc. IFIP TC6/WG 6.6 Symp. on Integrated Network Mngmt.*, pp. 101–109, Boston, MA, May 1989.

[2] G. Goldszmidt and Y. Yemini, "How to Build Manageable Systems: The Manager-Agent Delegation (MAD) Model," submitted to IFIP Integ. Network Mngmt. Sem., 1991.

[3] A. Dupuy, J. Schwartz, Y. Yemini, and D. Bacon, "NEST: A Network Simulation and Prototyping Testbed," *Comm. ACM 33*, vol. 10, pp. 63–74, Oct. 1990.

[4] M. T. Rose and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-Based Ethernets," Network Information Center, SRI International, Menlo Park, CA, May 1990.

[5] International Organization for Standardization, "Information Processing Systems—Open Systems Interconnection—Structure of Management Information," 1980.

[6] O. Wolfson, S. Sengupta, and Y. Yemini, "Modeling Network Management Functions as Database Operations," submitted to IFIP Integ. Network Mngmt. Sem., 1991.

Biography

Alexander Dupuy is currently a Senior Software Engineer in the Columbia University Computer Science Department. His current research interests are in network and systems management and object-oriented databases. He has extensive experience in managing networks based on Internet protocols.

Soumitra Sengupta is currently a Research Scientist in the Computer Science Department at Columbia University. He received his B.S. degree in electronics engineering from BITS, Pilani, India, in 1980, and Ph.D. in computer science from SUNY at Stony Brook in 1987. His current research interests include the design and development of large-scale software systems and network and application management.

Ouri Wolfson received the B.Sc. degree in mathematics from the University of Tel Aviv in 1976, and the Ph.D. degree in computer science from Courant Institute of Mathematical Sciences, New York University, in 1984. He is currently a Research Scientist in the Computer Science Department at Columbia University. Before joining Columbia, he was at American Broadcasting Co., AT&T Bell Laboratories, and the Technion. He is interested mainly in databases, communication network management, and rule processing, and he has published extensively in these areas.

Yechiam Yemini is an Associate Professor of Computer Science at Columbia University, where he founded and has directed the Distributed Computing and Communications (DCC) Laboratory. His research interests include tools for the design and management of distributed network systems, high-speed network protocols, performance analysis, and optimization of distributed systems. DCC Laboratory research resulted in widely distributed tools for network design and management used in academic and industrial research and development efforts. He was also a Founder and Director of Comverse Technology, a public U.S./Israeli company developing voice and fax store-and-forward systems.